Faculty Of Computers and Artificial Intelligence

Helwan University

Selected topics of cs II (CS396)

Multi- class classification of breast cancer abnormalities using Deep Convolutional Neural Network (CNN)

Team number: 9

1-Beshoy Ibrahim Asham

202000215 / 3rd Year / CS Major

2-Thaowpsta Saiid Aziz

202000233/ 3rd Year / CS Major

3-Ziad Mazhar Mahmoud

202000362/ 3rd Year / CS Major

4-Tassneam Mohsen Samy

202000224/ 3rd Year / CS Major

5-Batool Sherif Mohamed

202000195/ 3rd Year / CS Major

6- Ali Mohamed Mohamed

202000573 / 3rd Year / CS Major

Multi- class classification of breast cancer abnormalities using Deep Convolutional Neural Network (CNN)

TABLE OF CONTENTS

2) Paper details

a.<u>Author's name:</u>

- Maleika Heenaye-Mamode Khan
- Nazmeen Boodoo-Jahangeer
- Wasiimah Dullull
- Shaista Nathire
- Xiaohong Gao
- G. R. Sinha
- Kapil Kumar Nagwanshi

Paper's name:

Multi- class classification of breast cancer abnormalities using Deep Convolutional Neural Network (CNN)

Publisher's name:

Gulistan Raja

year of publication:

published: August 26, 2021

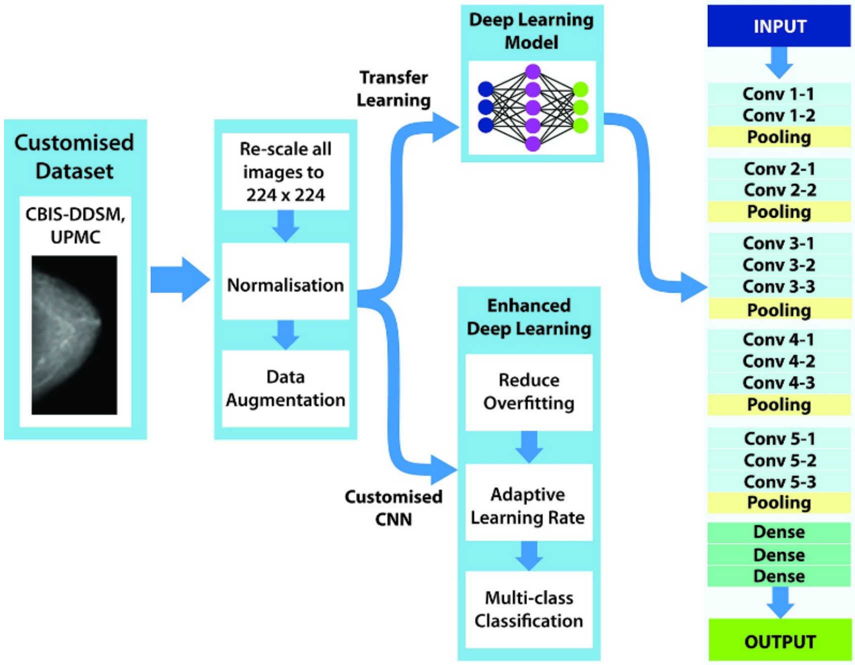
b. The dataset used:

Curated Breast Imaging Subset of Digital Database for Screening Mammography (CBIS-DDSM)

https://wiki.cancerimagingarchive.net/download/attachments/22516629/CBIS-DDSM-All-doiJNLP-zzWs5zfZ.tcia?version=1&modificationDate=1534787024127&api=v2

the implemented algorithms:

A deep convolution neural network (CNN) has been developed in addition to the application of an existing pre-trained deep learning model (RESNET50)



its results:

Model	Testing Accuracy (Overall)
RestNet50 Model	81.5%
Enhanced CNN Model	88%

https://doi.org/10.1371/journal.pone.0256500.t002

3) Project Description Document:

a. General Information on the selected dataset:

the name of the dataset used:

Breast Ultrasound Images Dataset (Dataset BUSI)

the link of dataset:

Breast Ultrasound Images Dataset (Dataset BUSI)

the total number of samples in the dataset: 5062

the dimension of images: 568x470

In case of classification

number of classes and their labels:

3 classes

- Normal
- Malignant
- Benign

b. Implementation details:

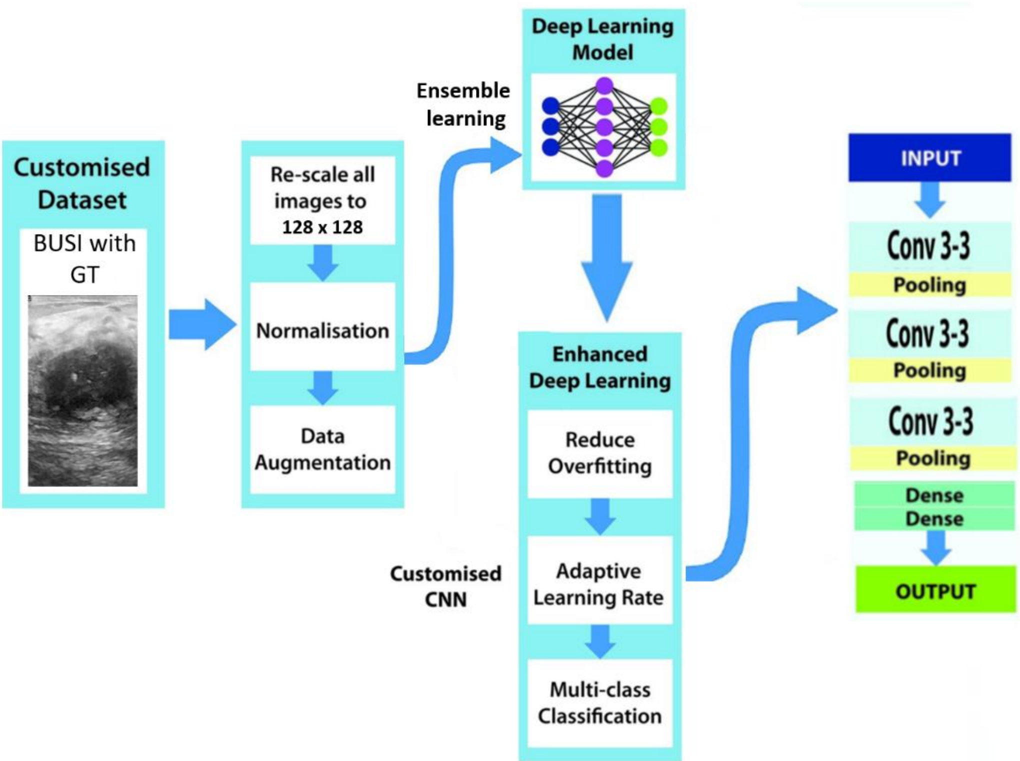
the ratio used for

- Training: 60%
- Validation: 20%
- Testing: 20%

the number of images in

- Training: 3038
- Validation: 1012
- Testing: 1012

A block diagram of the implemented model:



The hyperparameters used in the model:

- activation='relu','softmax'
- optimizer='adam'

```
    model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(img_size,img_size,3)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Conv2D(96, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(3, activation='softmax')
])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model



models = []
histories = []  # to store the training histories of each model
for i in range(num_models):
    # Create and compile model
    model = build_model()
    # Train model
    history = model.fit(train_generator.flow(train_x, train_y, batch_size=batch_size, shuffle=True),
                        epochs=epochs,
                        steps_per_epoch=train_x.shape[0] // batch_size,
                        validation_data=val_generator.flow(val_x, val_y, batch_size=batch_size, shuffle=True),
                        validation_steps=val_x.shape[0] // batch_size)
```

- loss='sparse_categorical_crossentropy'
- metrics=['accuracy']
- batch_size=32
- shuffle=True

<u>c. Results details:</u>

the measures that are used in evaluation and their results for the model:

F1 Score: 0.9855431762294384

Accuracy: 0.9851924975320829

Precision: 0.9854258308895405

Recall: 0.9857173334219939

Confusion matrix:

[347 9 1]

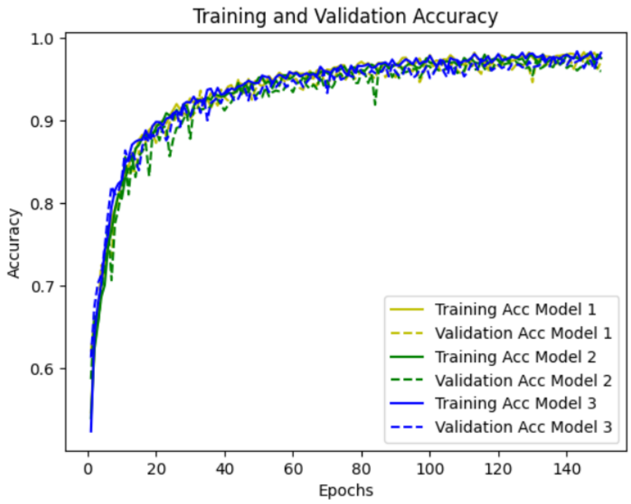
[5 332 0]

[0 0 319]

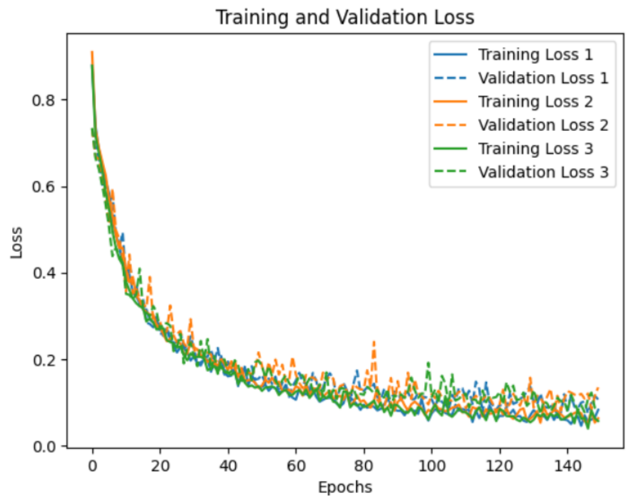
Average loss: 0.09040388216574986

```
32/32 [==============================]
32/32 [==============================]
32/32 [==============================]
F1 Score: 0.9855431762294384
Accuracy: 0.9851924975320829
Precision: 0.9854258308895405
Recall: 0.9857173334219939
Confusion matrix:
 [[347   9   1]
 [  5 332   0]
 [  0   0 319]]
32/32 [==============================]
32/32 [==============================]
32/32 [==============================]
Average loss: 0.09040388216574986
```

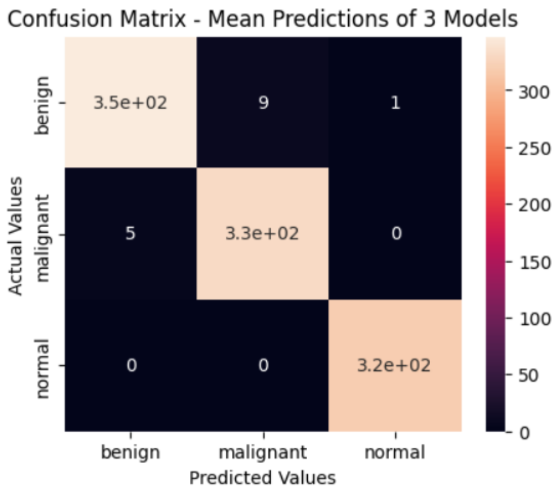
Training and validation accuracy



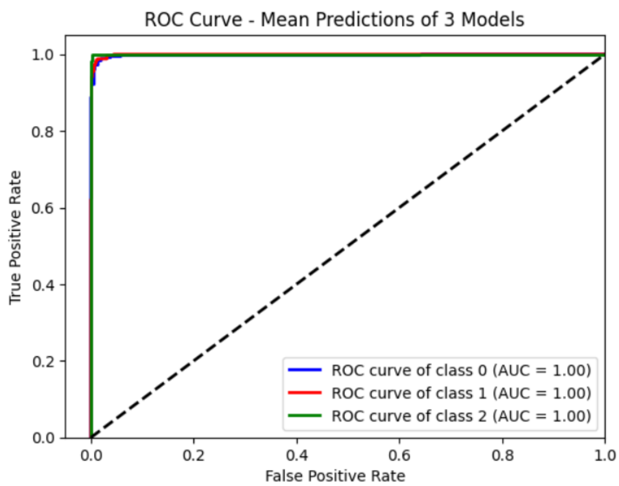
training and validation loss



Confusion matrix



ROC curve



Model Optimization:

We went through many stages in order to achieve high level of accuracy

-We first used the Transfer learning approach using pre-trained **ResNet50**

Which didn't end with good results

```
# Load pre-trained ResNet50 model
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(img_size, img_size, 3))

# Freeze the pre-trained layers
for layer in base_model.layers:
    layer.trainable = False

# Add custom top layers to the ResNet50 model
x = base_model.output
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(len(le.classes_), activation='softmax')(x)

# Create the custom model
model = Model(inputs=base_model.input, outputs=x)
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Set batch size
batch_size = 32

# Fitting the Model
history = model.fit(train_generator.flow(train_x, train_y, batch_size=32,shuffle=True),
                    epochs=50,
                    steps_per_epoch=train_x.shape[0] // batch_size,
                    validation_data=val_generator.flow(val_x, val_y, batch_size=32,shuffle=True),
                    validation_steps=val_x.shape[0] // batch_size)
```

during testing **(using 50 epochs),** the training accuracy that was achieved was **72.2%,** validation accuracy of **67.1%** and test accuracy was **74%**

```
- loss: 0.6316 - accuracy: 0.7220 - val_loss: 0.6048 - val_accuracy: 0.6719
```

```
Test loss: 0.5849946737289429
Test accuracy: 0.7405063509941101
```

-We tried adding the **CNN custom layered model** that was mentioned in the paper on top of the pre-trained **ResNet50 model**

```
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(img_size, img_size, 3))
# Freeze the pre-trained layers
for layer in base_model.layers:
    layer.trainable = False
# Add custom top layers to the ResNet50 model
x = base_model.output
x = tf.keras.layers.Conv2D(16,(1,1),activation = "relu", padding='same')(x)
x = tf.keras.layers.Conv2D(16,(3,3),activation = "relu", padding='same')(x)
x = tf.keras.layers.MaxPooling2D(2,2, padding='same')(x)
x = tf.keras.layers.Conv2D(32,(1,1),activation = "relu", padding='same')(x)
x = tf.keras.layers.Conv2D(32,(3,3),activation = "relu", padding='same')(x)
x = tf.keras.layers.MaxPooling2D(2,2, padding='same')(x)
x = tf.keras.layers.Conv2D(64,(1,1),activation = "relu", padding='same')(x)
x = tf.keras.layers.Conv2D(64,(3,3),activation = "relu", padding='same')(x)
x = tf.keras.layers.Conv2D(64,(1,1),activation = "relu", padding='same')(x)
x = tf.keras.layers.MaxPooling2D(2,2, padding='same')(x)
x = tf.keras.layers.Conv2D(128,(1,1),activation = "relu", padding='same')(x)
x = tf.keras.layers.Conv2D(128,(3,3),activation = "relu", padding='same')(x)
x = tf.keras.layers.Conv2D(128,(1,1),activation = "relu", padding='same')(x)
x = tf.keras.layers.MaxPooling2D(2,2, padding='same')(x)
x = tf.keras.layers.Conv2D(128,(1,1),activation = "relu", padding='same')(x)
x = tf.keras.layers.Conv2D(128,(3,3),activation = "relu", padding='same')(x)
x = tf.keras.layers.Conv2D(128,(1,1),activation = "relu", padding='same')(x)
x = tf.keras.layers.MaxPooling2D(2,2, padding='same')(x)
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(550, activation="relu")(x)
x = tf.keras.layers.Dropout(0.1, seed=2019)(x)
x = tf.keras.layers.Dense(400, activation="relu")(x)
x = tf.keras.layers.Dropout(0.3, seed=2019)(x)
x = tf.keras.layers.Dense(300, activation="relu")(x)
x = tf.keras.layers.Dropout(0.4, seed=2019)(x)
predictions = tf.keras.layers.Dense(len(np.unique(lbls_encoded)), activation="softmax")(x)
```

but it didn't seem to be working well with our dataset, the training accuracy that was achieved was **56.1%,** validation accuracy of **56.2%** and test accuracy was **56.3%**

```
loss: 5.7038 - accuracy: 0.5610 - val_loss: 5.7038 - val_accuracy: 0.5625
```

```
Test loss: 5.703782558441162
Test accuracy: 0.5632911324501038
```

-We decided to work on the **CNN model separately** as it seemed that the pre-trained ResNet50 model didn't fit well

In order to find a suitable CNN model for our dataset We decided to construct the layers, where we were given different accuracy percentages in each test which took us multiple times in order to reach this custom layered model and decide to work on it to achieve the desired accuracy.

```
# Define the model architecture
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(img_size,img_size,3)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(3, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Set batch size
batch_size = 32

# Fitting the Model
history = model.fit(train_generator.flow(train_x, train_y, batch_size=32,shuffle=True),
                    epochs=50,
                    steps_per_epoch=train_x.shape[0] // batch_size,
                    validation_data=val_generator.flow(val_x, val_y, batch_size=32,shuffle=True),
                    validation_steps=val_x.shape[0] // batch_size)
```

Initially, during testing **(using 50 epochs),** the training accuracy that was achieved was **66.6%,** validation accuracy of **58.6%** and test accuracy was **75%**

```
loss: 0.7042 - accuracy: 0.6667 - val_loss: 0.9092 - val_accuracy: 0.5868
```

```
Test loss: 0.7214857339859009
Test accuracy: 0.75
```

-We tried increasing the **epochs to 150**, the training accuracy that was achieved was **85.6%**, validation accuracy of **82.8%** and test accuracy was **85.4%**

```
- loss: 0.3382 - accuracy: 0.8569 - val_loss: 0.3069 - val_accuracy: 0.8281
```

```
Test loss: 0.3523615896701813
Test accuracy: 0.8544303774833679
```

The results were somehow good but not what we wanted.

-Finally, we decided to implement an **ensemble approach** that can help improve the accuracy of predictions by training the same CNN model thrice and combining their predictions, in order to capture a more diverse set of features in the data.

```
# Define the model architecture
def build_model():
    model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(img_size,img_size,3)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Conv2D(96, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(3, activation='softmax')
])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model


models = []
histories = []  # to store the training histories of each model
for i in range(num_models):
    # Create and compile model
    model = build_model()
    # Train model
    history = model.fit(train_generator.flow(train_x, train_y, batch_size=batch_size, shuffle=True),
                        epochs=epochs,
                        steps_per_epoch=train_x.shape[0] // batch_size,
                        validation_data=val_generator.flow(val_x, val_y, batch_size=batch_size, shuffle=True),
                        validation_steps=val_x.shape[0] // batch_size)
    # Add model and history to lists
    models.append(model)
    histories.append(history)
```

With **100 epochs** we achieved test accuracy of **96%** which seemed to be working pretty good and the accuracies indicate that the model is performing well and is not overfitting or underfitting.

```
F1 Score: 0.9619986405654859
Accuracy: 0.9615004935834156
```

-We tried increasing the **epochs to 150** using the same technique till we achieved the accuracy we were looking for which is **98.5%.**

```
F1 Score: 0.9855431762294384
Accuracy: 0.9851924975320829
```